Universidad de los Andes
Facultad de Ingeniería y Ciencias Aplicadas
Ciencias de la Computación

Sistemas Operativos y Redes

Tarea 3

Parte 2

Macarena Espinosa
Alfonso Zúñiga

A computer lets you make more mistakes faster than any invention in human history... with the possible exceptions of handguns and tequila.

Mitch Ratcliffe

Índice

1. Descripción General

Esta tarea consiste en diseñar un protocolo de mensajería instantánea basado en el intercambio de mensajes de texto plano y archivos entre personas, ya sea uno a uno o entre varias personas a la vez.

Para poder lograr esto, el protocolo debe asegurar que la entrega del mensaje será íntegro, es decir, el mensaje llegará a su destinatario tal como el usuario lo envía. Es por esto que decidimos diseñar un servicio orientado a la conexión y confiable, que tiene por objetivo principal la entrega completa del mensaje.

Por otro lado, con el fin de que la conexión entre hosts no se rompa o se vea afectada por cualquier irregularidad en el acceso a la red, utilizamos la arquitectura cliente-servidor, en la que todo mensaje enviado por un usuario llega al servidor, quien se encarga de hacerlos llegar a todos los receptores correspondientes.

Así, nuestro protocolo logra asegurar al usuario que su mensaje será entregado aunque el usuario destinatario no se encuentre conectado a la red en el momento o el envío del mensaje falle por razones externas. También asegura que la información será entregada en el orden original y en su total completitud.

A continuación, lo invitamos a leer de manera más específica las distintas partes de este protocolo y las razones que nos llevaron a ellas.

2. Tipos de hosts

Al momento de decidir qué tipo de host utilizaremos nos encontramos con dos grandes problemas:

- Suponemos un escenario en el que el destinatario no se encuentra conectado a la red. ¿Qué pasa con el mensaje? Dos opciones: o el mensaje nunca llega al destinatario, o el remitente tiene que intentar enviar el mensaje o archivo hasta que el destinatario se conecte.
- Por otro lado, tenemos que considerar la cantidad de conexiones que el usuario tendría que realizar si quiere enviar un mensaje o archivo a un grupo de contactos. Sin un intermediario, el usuario tendría que realizar tantas conexiones como destinatarios tiene el mensaje.

Debido a todo esto decidimos implementar el modelo cliente-servidor, en el que el usuario remitente envía el mensaje al servidor, y este se encarga de hacer llegar la información a todos los destinatarios apenas estos se encuentren conectados a la red, quitándole la carga al usuario.

3. Formato de mensajes

El mensaje enviado al servidor es codificado utilizando *Protobuf* [1], un mecanismo de serialización de estructuras de datos desarrollado por Google, que recibe instancias de una clase definida dentro de un archivo `.proto`. En de los métodos entregados por esta aplicación se encuentra la serialización, que utilizaremos para adaptar el mensaje junto con toda la información de destino antes de ser enviado al servidor.

La implementación de este servicio exigiría al usuario instalar *Protobuff* en sus equipos. La necesidad de herramientas secundarias para la implementación de software es una practica muy común en el desarrollo y distribución de software. Normalmente el archivo de instalación de estos incluyen todos los complementos necesarios para su funcionamiento, pero éste no sera el caso para nuestra entrega, al no ser necesario un archivo de instalación; se deja como responsabilidad del usuario procurar instalar *Protobuff*.

A continuación se presenta la clase que se pretende utilizar para detallar la información del segmento que sera enviado por los sockets posterior a su codificación por parte de *Protobuf*:

```
1 class Segment:
2     def __init__(self, group, ip_destination, ip_sender, content):
3         self.group = group
4         self.ip_destination = ip_destination
5         self.ip_sender = ip_sender
6         self.data_type = data_type
7         self.content = content
8         self.date = date
```

- **group**: Booleano que indica si el ip-destination es la IP del receptor o la ID de un chat grupal que el servidor debe buscar en sus registros.
- **ip-destination**: La dirección IP del destino o ID del chat grupal, según el valor de **group**.
- **ip-sender**: La IP desde donde originalmente se envió el mensaje.
- **data-type**: Booleano que indica si se trata de un mensaje (True) o un archivo (False).
- **content**: El contenido del mensaje.
- **date**: Fecha y hora en el que el mensaje fue enviado.

[1] Para mayor información, ingrese a https://developers.google.com/protocol-buffers/

4. Comunicación

En cuanto al tipo de servicio, decidimos que un servicio orientado a la conexión es lo más apto, debido a que nos asegura que al receptor recibirá todo el contenido del mensaje enviado por el emisor.

También tuvimos que tomar la decisión sobre cuál modo de transferencia utilizaríamos. Como esto se trata de envío de mensajes y archivos en un momento específico y no un constante envío de contenido, preferimos la transferencia de modo mensaje por sobre el modo streaming.

En la siguiente figura se muestra cómo se relacionan las diferentes partes. Primero, el emisor 'saluda' al servidor, y éste le confirma la conexión. Luego el emisor le envía el mensaje al servidor y termina la conexión. Este mismo proceso se repite entre el servidor y el verdadero destinatario del mensaje. En caso de que el mensaje vaya a un grupo, el servidor repite esto con todos los contactos pertenecientes a tal grupo.

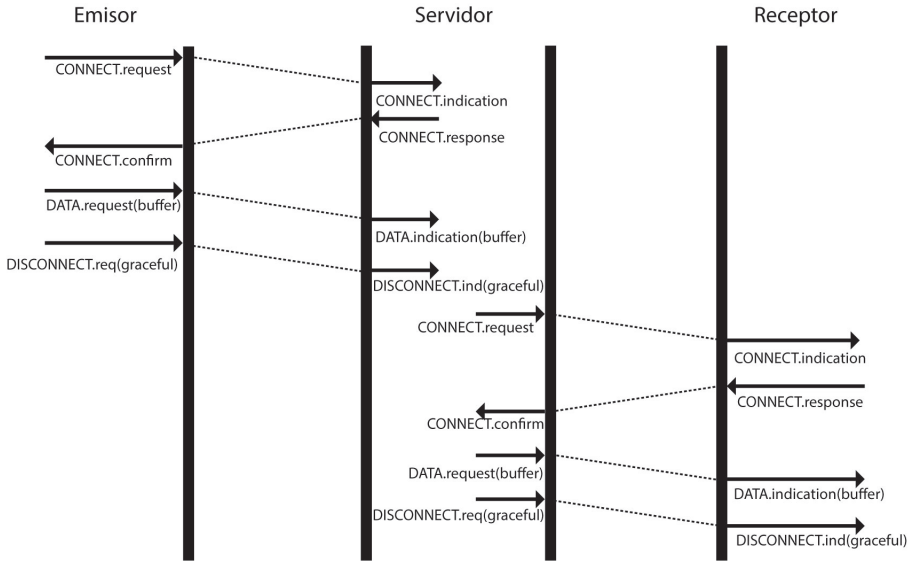

Figura 1: Servicio orientado a la conexión

Si hay un error en el envío del mensaje debido a un problema en uno de los host, éste terminará la conexión de manera abrupta, como se muestra en la siguiente figura. En este caso, el host que envía el mensaje volverá a conectarse con el destinatario y el proceso se repetirá hasta que el mensaje sea enviado de manera correcta y completa.

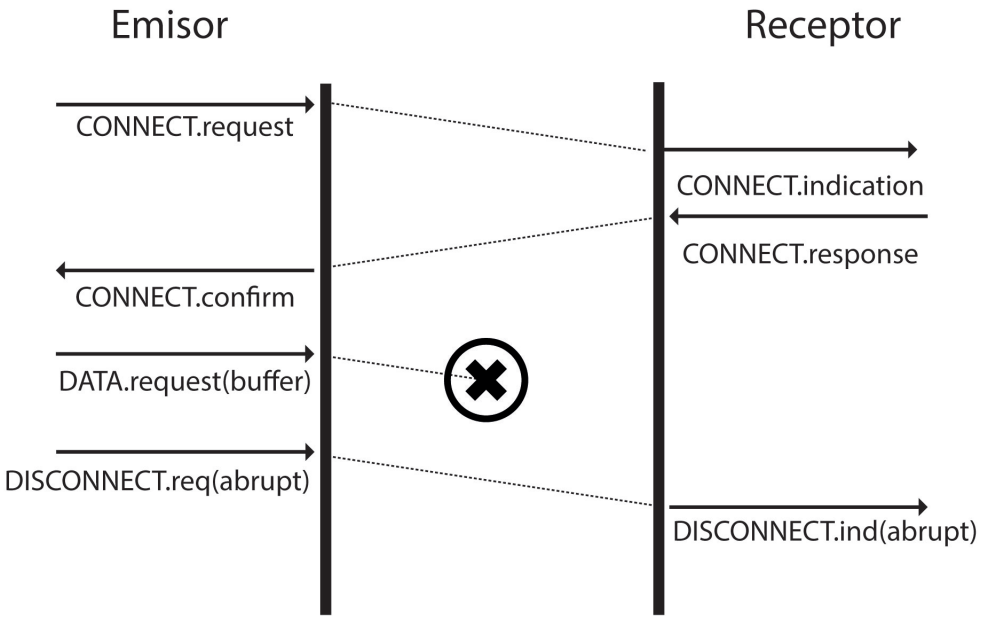

Figura 2: Ejemplo de error en la conexión

Pero antes que todo esto, el usuario se debe registrar en en el servicio para que el servidor sepa quién es y dónde está. Sólo después de realizar esta acción es que el usuario puede enviar un mensaje o recibir uno.

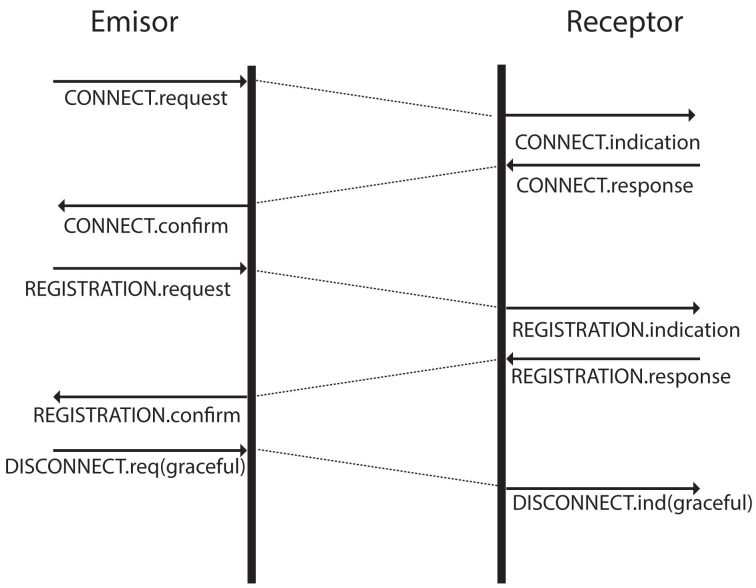

Figura 3: Ejemplo de registro de usuario

En el caso de que ocurra un error (por ejemplo, cliente ya registrado), el servidor enviará un REGISTRATION.error especificando la razón del error, dejando en manos del usuario si decide volver a intentarlo o desconectarse.

5. Interacciones

El funcionamiento de este sistema comienza con un usuario escribiendo un mensaje o adjuntando un archivo. Cuando se disponga a enviar, este pasará a ser instanciado en un archivo `.proto`, para luego ser serializado a través de *Protobuf*, dejando la información lista para ser enviada al servidor a través de sockets.

Dentro del servidor de decodificará el mensaje (también utilizando la lógica que proporciona *Protobuf*) para conocer la ruta del destinatario; procederá a intentar abrir una conexión con éste y enviar el mensaje. En case de no poder establecer una conexión, el mensaje se almacenará en el servidor hasta que se logre establecer la conexión. En el caso de que el mensaje fuera direccionado a un grupo de receptores, el servidor buscará el identificador del grupo especifico contenido en sus registros y conseguirá las direcciones de todos los integrantes, abriendo una conexión con cada uno de ellos (excepto con el emisor) y procede a enviar el mensaje a cada uno. En caso de no lograr establecer una conexión con alguno de ellos, se procederá de igual manera como se explicó antes.

Cuando el mensaje es entregado al receptor, éste se decodifica y se alerta al receptor, desplegando el nombre del emisor (si es que esa información esta disponible) y la opción de ver el mensaje.

Luego de la llegada de la totalidad del mensaje, la conexión con el servidor se cierra, solo para volver a abrirse en el caso que el destinatario procediera a responder con un nuevo mensaje, invirtiendo los roles y comenzando el proceso todo de nuevo.

6. Estados

A continuación se detallan los diferentes estados en los que se pueden encontrar los distintos hosts en el transcurso de una interacción entre dos o varios de ellos, detallando que ocurre en caso de errores, imprevistos o funcionamiento ideal.

6.1. Clientes

- **Idle:** El usuario se encuentra desconectado cuando no esta enviando ni recibiendo mensajes, más allá de si se encuentra conectado a la red de Internet o

no.

- **Connecting:** El usuario se está registrando con el servidor para iniciar una conexión. Si no se logra establecer una conexión con el servidor, el usuario recibe una notificación de error explicando que no se pudo realizar la operación.
- **Sending:** El usuario se encuentra conectado al servidor. Este estado llega a su fin una vez que el mensaje haya sido enviado al servidor y éste envié la señal de confirmación para informar al usuario que todo el contenido le ha llegado.
- **Listening:** el usuario pasa a estar conectado, pero ahora sucede cuando el servidor intenta establecer una conexión con el receptor. Al momento de lograr crear este enlace, el servidor se encarga de enviar los paquetes de información al receptor. La conexión se cierra cuando el servidor recibe la señal de que el receptor ha recibido toda la información enviada.
- **Delivery Failure:** Falla en recepción o envío de mensaje.

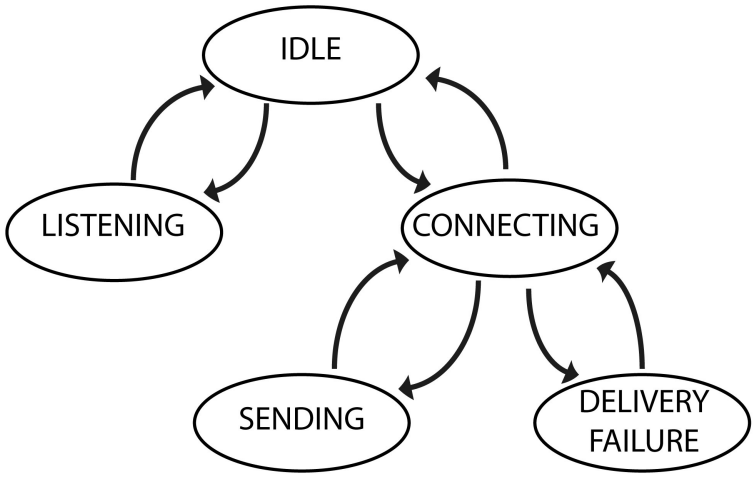

Figura 4: Máquina de estados del cliente

6.2. Servidor Central

El servidor puede presentarse en varios estados con respecto a la conexión con un usuario. Si bien este siempre esta operando a la espera de llamados por parte de los usuarios, hablaremos sobre su estado respecta a su interacción con un usuario especifico **u**.

- **Idle:** El servidor no esta realizando interacción alguna con el usuario **u**.
- **Client Registration:** llega al servidor una solicitud de usuario **u**, que esta intentando instanciar una conexión con el servidor para enviar un mensaje.
- **Client Meesage Reception:** el mensaje llega al servidor y este lo decodifica (utilizando la lógica de *Protobuf*) para conocer la dirección de destino. Si el mensaje es grupal, pasará a estado "Group Message Delivery"; si no, pasa directo al estado Çlient Message Delivery".
- **Group Message Delivery:** Si el mensaje recibido por el servidor es de naturaleza grupal, el host se encarga de buscar dentro de sus registros por el grupo en cuestión, cuyo ID fue entregado dentro de la codificación del mensaje (en lugar de la IP de destino). Una vez encontrado el grupo, se extrae un arreglo con todas las IP's de sus integrantes, y el servidor pasa a estado Çonectando".
- **Client Message Delivery:** El servidor envía la solicitud de conexión a el/los usuario(s) **u** para enviar un mensaje. Una vez que el/los usuario(s) aceptan esta solicitud, procede a enviar los paquetes de datos, solo pasando a estado Ïdleüna vez que recibe confirmación de que todos los datos han sido enviados.

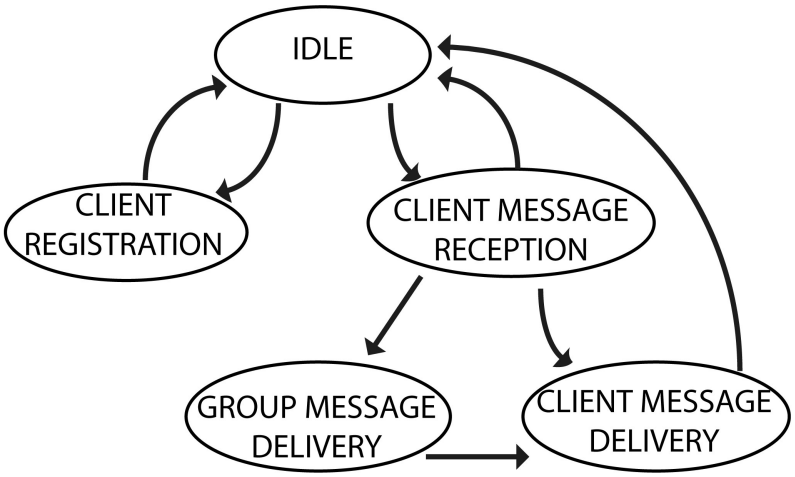

Figura 5: Máquina de estados del servidor